We claim:

1. A system for reusing profile information of a program, comprising:

a comparator to define a match when a first value equals a second value;

a propagator to propagate profile information when a match is defined; and

a processing engine that processes a portion of a first version of the program to produce the first value and a portion of a second version of the program to produce the second value, wherein the processing engine uses a set of information at a desired fuzziness level to produce the first value and the second value.

2. The system of claim 1, wherein the processing engine includes an engine to produce values for procedures.

3. The system of claim 1, wherein the processing engine includes an engine to produce values for portions of data.

4. The system of claim 1, wherein the processing engine includes an engine to produce values for portions of code within the procedures.

5. The system of claim 1, further comprising an annotator to annotate the strength of the match so as to enhance the propagation of profile information.

6. A method for comparing versions of a program in binary format, comprising:

finding equivalent contents in portions of two versions of the program;

finding equivalent structure in the portions of the two versions; and

forming a match when a portion of one of the two versions is an equivalence to a portion of the other of the two versions.

7. The method of claim 6, wherein the act of finding equivalent contents includes finding equivalent code in the portion of one version of the two versions and the portion of the other version of the two versions.

8. The method of claim 7, wherein the act of finding equivalent contents includes finding equivalent data in the portion of one version of the two versions and the portion of the other version of the two versions.

9. The method of claim 6, wherein the act of forming a match includes forming a match based on a fuzzy match for portions of the two versions that include small differences.

10. The method of claim 6, wherein the act of forming a match includes forming a match based on the results of both the act of finding equivalent contents and the

act of finding equivalent structure.

11. A computer readable medium having instructions stored thereon for causing a computer to perform a method for comparing versions of a program in binary format, the method comprising:

finding equivalent contents in portions of two versions of the program;

finding equivalent structure in the portions of the two versions; and

forming a match when a portion of one of the two versions is an equivalence to a portion of the other of the two versions.

12. The method of claim 11, wherein the act of finding equivalent contents includes finding equivalent code in the portion of one version of the two versions and the portion of the other version of the two versions.

13. The method of claim 12, wherein the act of finding equivalent contents includes finding equivalent data in the portion of one version of the two versions and the portion of the other version of the two versions.

14. The method of claim 11, wherein the act of forming a match includes forming a match based on a fuzzy match for portions of the two versions that include small differences.

15. The method of claim 11, wherein the act of forming a match includes forming a match based on the results of both the act of finding equivalent contents and the act of finding equivalent structure.

16. A method for comparing versions of a program in binary format, comprising:

finding equivalent procedures in a first version and a second version;

finding equivalent portions of data in equivalent procedures; and

finding equivalent portions of code in equivalent procedures.

17. The method of claim 16, wherein the act of finding equivalent procedures includes comparing information selected from a group consisting of name information, type information, and code information.

18. The method of claim 16, wherein the act of finding equivalent portions of code includes finding equivalent portions of code based on a hash value, which is calculated from the instructions in the portions of code and the ordering of the instructions.

19. The method of claim 18, wherein the act of finding equivalent portions of code based on a hash value includes iterating the act of finding with different levels of matching fuzziness so as to enhance the act of finding equivalent portions of code.

20. The method of claim 16, wherein the method executes a desired combination of at least one of the act of finding the equivalent procedure, the act of finding equivalent portions of data, and the act of finding equivalent portions of code.

21. A method for comparing procedures in versions of a program, comprising:

finding procedures having identical names;

finding procedures by calculating one hash value based on the code;

finding procedures having similar names; and

finding procedures by comparing hash values for portions of procedures.

22. The method of claim 21, wherein the method executes a desired combination of acts, wherein the desired combination includes at least one of the act of finding procedures having identical names, the act of finding procedures by calculating a hash value based on the code, the act of finding procedures having similar names, and the act of finding procedures by comparing hash values for portions of procedures.

23. The method of claim 21, wherein the act of finding procedures having identical names includes executing at least one of an act of finding procedures having identical extended names and an act of finding procedures having identical hierarchical names.

24. The method of claim 21, wherein the act of finding procedures by calculating a hash value based on the code includes finding procedures that lack identical names by calculating the hash value that considers the order of the code in the procedures.

25. The method of claim 21, wherein the act of finding procedures having similar names includes finding procedures having small differences in the hierarchical names and having a high percentage of blocks between the procedures that have equivalent hash values.

26. The method of claim 21, wherein the act of finding procedures by comparing hash values for portions of procedures includes finding procedures, which cannot be matched by names, by comparing whether a hash value for blocks within one procedure is equivalent to a hash value for blocks within another procedure.

27. A method for comparing data in a first version to a second version of a program in binary format, comprising:

finding equivalent portions of data using hash values; and

finding equivalent portions of data using positional information in the procedures.

28. The method of claim 27, wherein the method executes a desired combination of acts, wherein the desired combination includes at least one of the act of finding equivalent portions of data using hash values and the act of finding equivalent portions of data using positional information in the procedures.

29. The method of claim 27, wherein finding equivalent portions of data using positional information includes defining that a portion of data in the first version matches a portion of data in the second version when the portion of data sandwiches between previously matched portions of data.

30. A method for comparing code in a procedure in a first version to a procedure in a second version of a program in binary format, comprising:

finding equivalent portions of code using hash values; and

finding equivalent portions of code using control flow.

31. The method of claim 30, wherein the method executes a desired combination of acts, wherein the desired combination includes at least one of the act of finding equivalent portions of code using hash values and the act of finding equivalent portions of code using control flow.

32. The method of claim 30, wherein the act of finding equivalent portions of code using hash values includes calculating hash values based on code contents and

positional information.

33. The method of claim 30, wherein the act of finding equivalent portions of code using hash values is iterated, wherein each iteration uses a desired level of fuzziness to define a match between a portion of code in the procedure in the first version and a portion of code in the procedure in the second version of code.

34. The method of claim 30, wherein the act of finding equivalent portions of code using control flow selected from a group consisting of conditional branches, jump instructions, return instructions, and previously matched portions of code so as to define that a portion of code in the procedure in the first version matches a portion of code in the procedure in the second version.

35. A method for hashing code to compare versions of a program, comprising:

defining an instruction that contains an operand, which appears in a control flow, as a source instruction, and defining the block referred to by the operand as a target block; and

forming a hash value, wherein forming includes calculating the hash value by treating the operand in a block in one version of the program the same as the operand in a block in the other version of the program when the target block has previously been matched to another block.

36. The method of claim 35, wherein forming includes formulating the hash value based on the extended name if the target block already has a compiler-assigned extended name.

37. The method of claim 36, wherein forming includes figuring the hash value based on the offset of the address of the target block from the beginning of a procedure and the offset of the address of the target block from the source instruction when the procedure contains the target block and the source instruction.

38. The method of claim 37, wherein forming includes determining the hash value based on the name of a procedure and the offset of the address of the target block from the beginning of the procedure when the procedure that contains the target block is different from another procedure that contains the source instruction.

39. The method of claim 38, wherein the method selectively executes the act of calculating, the act of formulating, the act of figuring, and the act of determining so as to inhibit errors from arising in forming the hash value.

40. A method for comparing code in versions of a program, comprising:

hashing to form hash values from a set of information at a desired level; and

comparing the hash value of a portion of code in the first version to the hash value of a portion of code in the second version so as to define a match if the hash

value of the portion of code in the first version equals the hash value of the portion of code in the second version.

41. The method of claim 40, further comprising iterating the method, wherein iterating includes iterating the act of hashing to form hash values from another set of information of another desired level.

42. The method of claim 40, wherein comparing includes comparing the hash value of a procedure in the first version to the hash value of a procedure in the second version, wherein the act of comparing defines a match by the order of appearance of procedures in the program when multiple procedures have the same hash value.

43. The method of claim 42, wherein comparing includes comparing the hash value of each portion of code in a procedure in the first version to the hash value of each portion of code in a procedure in the second version, wherein the act of comparing defines a match when a desired percentage of portions of code in the procedure in the first version matches the portions of code in the second version.

44. The method of claim 40, wherein comparing includes comparing the hash value of a portion of code in a procedure in the first version to the hash value of a portion of code in a procedure in the second version, wherein the act of comparing

executes a two-phase process when multiple portions of code have the same hash value.

45. The method of claim 40, wherein comparing includes executing a two-phase process, wherein the first phase of the two-phase process includes forming a match that selectively limits cross-matching, and wherein the second phase of the two-phase process includes propagating a match between two portions of code to other portions of code in the vicinity of the two portions of code using a neighbor test.

46. The method of claim 40, further comprising annotating each match to form information for subsequent analysis.